

**Instituto Superior de Engenharia de Lisboa**

Licenciatura em Engenharia Informática e Multimédia

Semestre de inverno 2019/2020

# **Redes de Computador**

## **Trabalho Prático I**



**Trabalho elaborado pelo Grupo 6:**

[Redacted student names]

**Docente:** Luís Pires

[Redacted signature]

## Índice

1 – Introdução .....	4
1.1 – Modelo OSI.....	5
1.2 – Protocolo HTTP .....	7
2 – Desenvolvimento .....	9
2.1 Código 200.....	10
2.2 Código 302.....	12
2.3 Código 404.....	14
2.4 Código 500.....	16
3 – Conclusões .....	18
4 – Bibliografia .....	19

## Índice de figuras

Figura 1 - Camadas do Modelo OSI .....	5
Figura 2 - Funcionamento do modelo OSI numa analogia aos correios .....	6
Figura 3 - Exemplo de uma resposta HTTP .....	7
Figura 4 - Formato da resposta HTTP .....	8
Figura 5 - Wireshark em ambiente gráfico no sistema operativo Ubuntu .....	9
Figura 6 - Pacotes HTTP capturados do estado 200 .....	10
Figura 7 - Informação do pedido do estado 200, como capturado pelo Wireshark .....	10
Figura 8 - Informação da resposta do estado 200, como capturado pelo Wireshark .....	10
Figura 9 - Pacotes HTTP capturados do estado 20, com pedidos feitos através do cliente Java.....	11
Figura 10 - Informação do pedido do estado 200, como capturado pelo Wireshark,, através do cliente Java.....	11
Figura 11 - Informação do pedido pelo cliente .....	11
Figura 12 - Informação da resposta do estado 200, como capturado pelo Wireshark, através de pedidos feitos pelo cliente Java.....	11
Figura 13 - Informação da resposta data pelo cliente .....	11
Figura 14 - Informação do pedido do estado 302, como capturado pelo Wireshark .....	12
Figura 15 - Pacotes HTTP capturados do estado 302 .....	12
Figura 16 - Informação da resposta do estado 200, como capturado pelo Wireshark .....	12
Figura 17 - Pacotes HTTP capturados do estado 302, com pedidos feitos através do cliente Java...	13
Figura 18 - Informação do pedido do estado 302, como capturado pelo Wireshark, através do cliente Java.....	13
Figura 19 - Informação do pedido pelo cliente .....	13
Figura 20 - Informação da resposta do estado 302, como capturado pelo Wireshark, através de pedidos feitos pelo cliente Java.....	13
Figura 21 - Informação da resposta data pelo cliente .....	13
Figura 22 - Pacotes HTTP capturados do estado 404 .....	14
Figura 23 - Informação do pedido do estado 404, como capturado pelo Wireshark .....	14
Figura 24 - Informação da resposta do estado 404, como capturado pelo Wireshark .....	14
Figura 25 - Pacotes HTTP capturados do estado 404, com pedidos feitos através do cliente Java...	15
Figura 26 - Informação do pedido do estado 404, como capturado pelo Wireshark, através do cliente Java.....	15
Figura 27 - Informação do pedido pelo cliente .....	15
Figura 28 - Informação da resposta do estado 404, como capturado pelo Wireshark, através de pedidos feitos pelo cliente Java.....	15
Figura 29 - Informação da resposta data pelo cliente .....	15
Figura 30 - Informação do pedido do estado 500, como capturado pelo Wireshark .....	16
Figura 31 - Pacotes HTTP capturados do estado 500 .....	16
Figura 32 - Informação da resposta do estado 500, como capturado pelo Wireshark .....	16
Figura 33 - Pacotes HTTP capturados do estado 500, com pedidos feitos através do cliente Java...	17
Figura 34 - Informação do pedido do estado 500, como capturado pelo Wireshark, através do cliente Java.....	17
Figura 35 - Informação do pedido pelo cliente .....	17
Figura 36 - Informação da resposta data pelo cliente .....	17
Figura 37 - Informação da resposta do estado 500, como capturado pelo Wireshark, através de pedidos feitos pelo cliente Java.....	17

## Lista de Acrónimos

DARPA: Defense Advanced Research Projects Agency, 5

HTTP. *Hypertext Transfer Protocol*

IP: Internet Protocol, 5

ISO. International Organization for Standardization

MAC. Media Access Control

MIME. *Multipurpose Internet Mail Extensions*

Modelo OSI: Open Systems Interconnection, 5

RFC. *Request for Comments*

TCP: Transmission Control Protocol, 5

URI. Uniform Resource Identifier

URL: Uniform Resource Locator, 7

VPN: Virtual Private Network, 9

# 1 – Introdução

Nesta primeira fase do trabalho pretende-se o estudo e análise de mensagens de HTTP, com auxílio à ferramenta *Wireshark*, trocadas entre servidor e cliente em computadores distintos. É pretendido ainda que a troca de mensagens seja feita por dois tipos de clientes, um cliente browser e um cliente web implementado com auxílio a uma linguagem de programação à escolha.

## 1.1 – Modelo OSI

Existem dois modelos de redes de computadores: O modelo TCP/IP e o Modelo OSI. O modelo TCP/IP é um modelo que se divide em 4 camadas, desenvolvido pela DARPA, em 1969. Enquanto que o modelo TCP/IP é muito ligado aos protocolos, ou seja, as camadas já definem que protocolos usam, o Modelo OSI apresenta-se como mais abstrato. O modelo lecionado é o Modelo OSI, que é onde nós nos vamos focar.

O Modelo OSI é um modelo de redes de Computadores criado em 1971, que é um dos modelos *standard* na arquitetura de redes de comunicação entre computadores. Este modelo foi desenvolvido pela ISO (*International Organization for Standardization*).

Este modelo divide-se em 7 camadas, onde cada camada contém um conjunto de protocolos que operam ao nível da camada.

As 7 camadas do modelo OSI são:

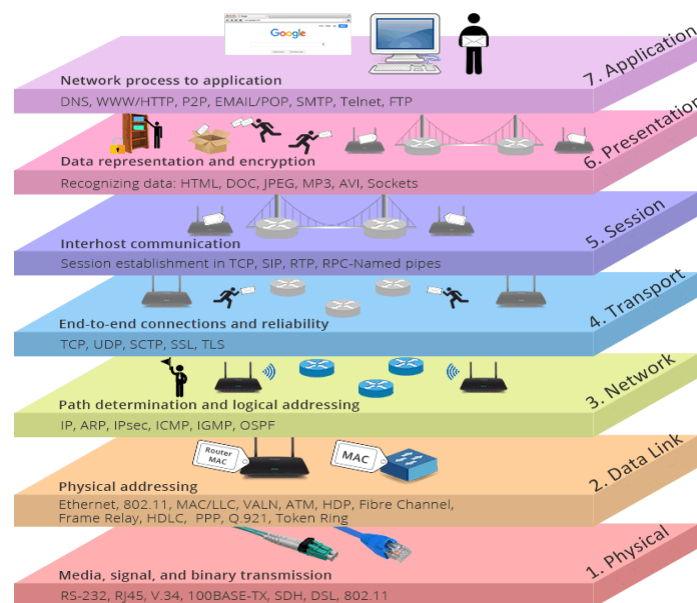


Figura 1 - Camadas do Modelo OSI

As funções de cada camada podem ser resumidas a: - A **camada Física** é responsável pelos meios de transporte dos dados, como pelos equipamentos e protocolos que operam nesta camada.

A **camada de ligação de dados** é a camada que controla o acesso aos meios de transmissão, e que também corrige possíveis erros, antes de enviar os dados para a camada física. Esta camada sabe converter IP's para endereços MAC (O endereço físico da máquina)

A **camada de Rede** é a camada que faz o encaminhamento de pacotes e também sabe endereçar os pacotes a determinados *hosts*. Esta camada sabe identificar o IP (endereço lógico) da máquina, e determina qual o melhor caminho para chegar à máquina-destino do pedido

A **camada de Transporte** é onde funcionam os protocolos que garantem o envio dos pacotes de dados. Na generalidade, esta camada controla o fluxo de informação e faz também controlo de erros

A **Camada de Sessão** gere (e também estabelece, e termina) sessões entre as aplicações. Por exemplo, numa arquitetura Cliente-Servidor, esta camada estabelecerá a sessão entre um Browser e o Servidor Web, e iria gerir essa sessão.

A **camada de apresentação** é a camada que codifica os dados, e assegura a compatibilidade entre as aplicações de sistema diferentes

A **camada de aplicação** é onde estão todas as aplicações a correr, que dão início à comunicação. Esta camada utiliza protocolos que outras aplicações também sabem interpretar, e que são padrões em termos de Internet. A nível de camadas do Modelo OSI, o protocolo HTTP situa-se na nesta camada, que é a camada dá início à criação do pacote(s) de dados a serem enviados.

Na prática, o funcionamento pode ser resumido ao envio de uma encomenda

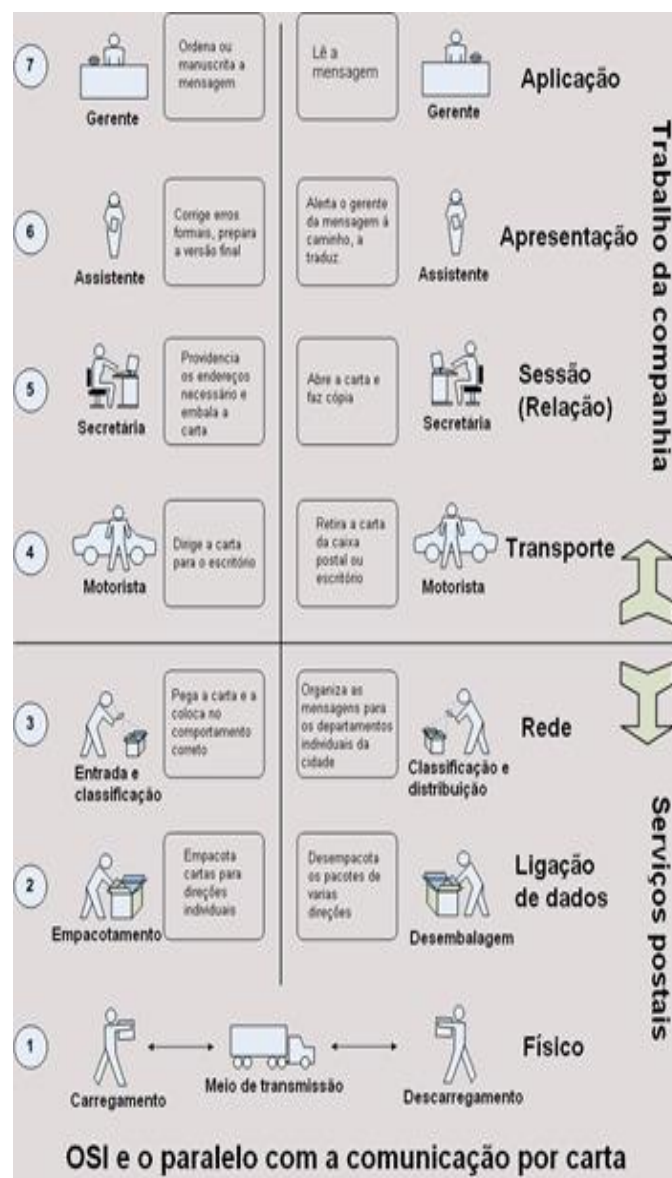


Figura 2 - Funcionamento do modelo OSI numa analogia aos correios

## 1.2 – Protocolo HTTP

O HTTP (*Hypertext Transfer Protocol*) é um protocolo de comunicação utilizado na Web para sistemas de informação de hipermídia distribuídos. Este protocolo assenta no modelo cliente-servidor, onde são trocadas mensagens entre ambos através de *sockets*, com auxílio ao protocolo TCP para transporte dos dados. Este protocolo é *stateless* (não mantém estado), podendo mesmo assim utilizar alguns métodos para que isso seja possível como variáveis ocultas dentro de formulários, Cookies ou parâmetros de *query string*. A sua ligação pode ser persistente, onde as ligações permanecem abertas após o envio das respostas, ou não persistente, onde cada objeto requisitado pelo cliente é transportado por utilização TCP provocando muitas ligações paralelas.

As requisições e respostas HTTP compartilham uma estrutura similar e são compostas por:

1. Uma linha inicial (*start-line*) que descreve as requisições a serem implementadas, ou o seu estado de sucesso ou falha;
2. Um conjunto opcional de cabeçalhos HTTP especificando a requisição, ou descrevendo o corpo incluído na mensagem;
3. Uma linha em branco (*empty line*) indicando que toda a meta-informação já foi enviada;
4. Um corpo (*body*) onde contém os dados associados à requisição, ou um documento associado à resposta. A presença do corpo e o seu tamanho são especificados pela linha inicial e pelos cabeçalhos HTTP.

Quanto às mensagens de *request* estas podem ser de vários tipos:

- **GET** – Para obtenção de dados. Solicita a representação de um recurso específico;
- **HEAD** – Solicita uma resposta idêntica ao método GET, mas deixa o objeto fora da resposta;
- **POST** – Para envio de dados. Submete uma entidade a um recurso específico, causando mudança de estado;
- **PUT** – Substitui todas as atuais representações do recurso de destino pelos dados da requisição (informação contida no campo URL);
- **DELETE** – Remove um recurso específico indicado no campo URL.

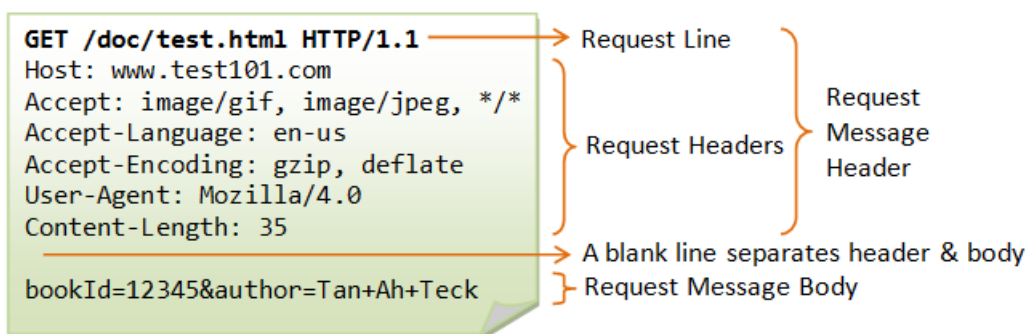


Figura 3 - Exemplo de uma resposta HTTP

Quanto às mensagens de resposta estas podem conter vários códigos de estado:

- 1xx (Informação) – Utilizada para enviar informações para o cliente de que a sua requisição foi recebida e está a ser processada;
- 2xx (Sucesso) – Indica que a requisição do cliente foi bem-sucedida;
- 3xx (Redirecionamento) – Informa a ação adicional que dever ser tomada para completar a requisição;
- 4xx (Erro no cliente) – Avisa que o cliente fez uma requisição que não pode ser atendida;
- 5xx (Erro no servidor) – Ocorreu um erro no servidor ao cumprir uma requisição válida

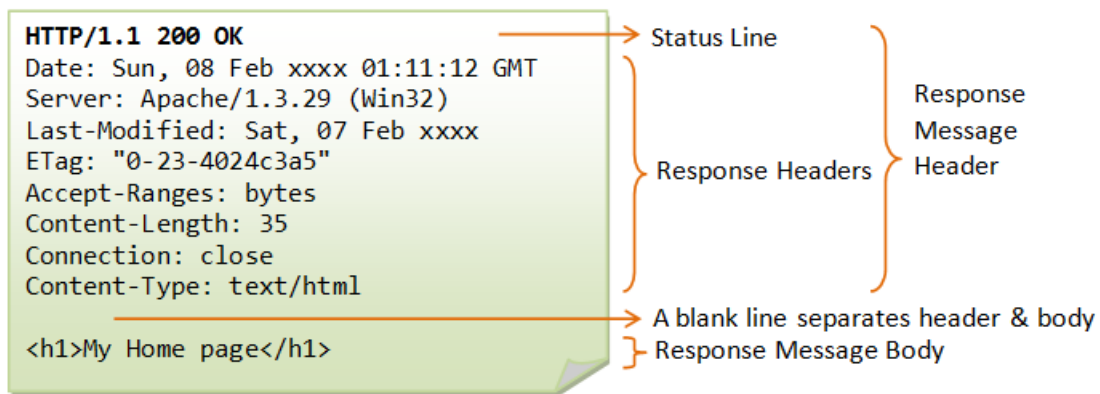


Figura 4 - Formato da resposta HTTP

Neste relatório vamos então demonstrar como os conceitos teóricos lecionados durante as aulas se relacionam com a Prática. Para isso, vamos usar algumas ferramentas que nos permitem aplicar e validar nosso conhecimento teórico.



## 2 – Desenvolvimento

Para analisar os pacotes dados, e ver que protocolos usados durante a comunicação entre máquinas, vamos usar o *software* Wireshark, e para criar um servidor Web vamos usar o *software* Apache. Mais à frente, vamos demonstrar um pequeno cliente que corre em modo consola, que foi desenvolvido com recurso à linguagem Java.

Como referido na introdução, para a analisar o tráfego de rede, vamos utilizar o *software* Wireshark. Este programa, entre outras funcionalidades, permite ver o conteúdo dos pacotes de dados, tal organizar os pedidos por protocolo [1], bem como a aplicação de filtros, que nos permitem por exemplo, filtrar tráfego por protocolo ou por host, através de uma interface gráfica.

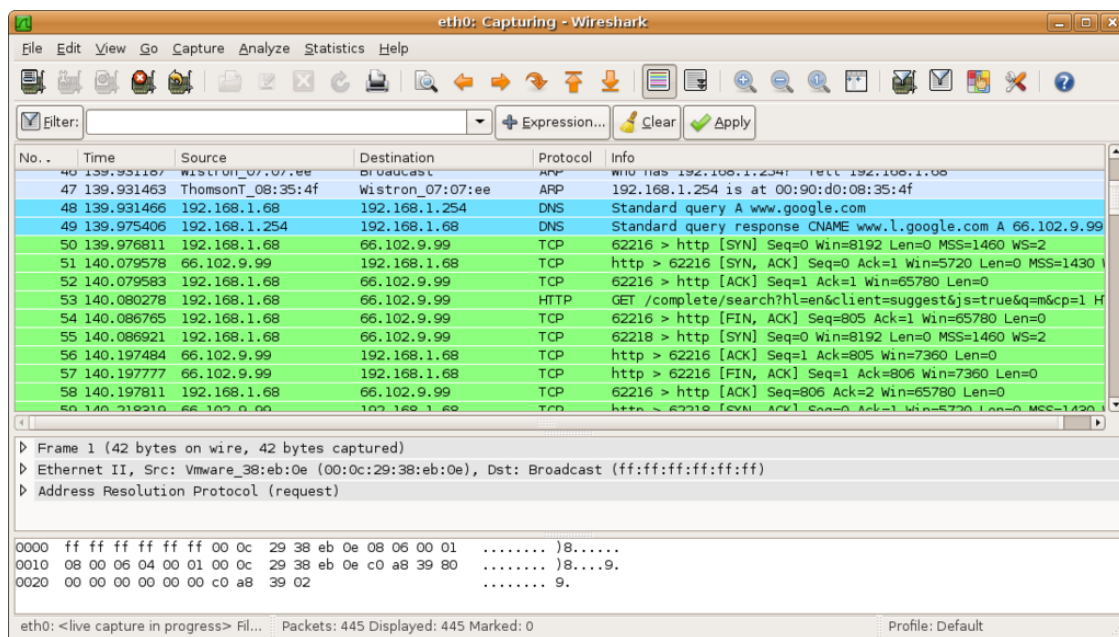


Figura 5 - Wireshark em ambiente gráfico no sistema operativo Ubuntu

Outras alternativas a este *software* seria o *tcpdump*, que corre em linha de comandos. Neste relatório, vamos utilizar as funcionalidades referidas acima para analisar os pacotes de dados capturados.

Nesta parte de desenvolvimento do trabalho utilizámos o servidor web Apache e dois clientes para trocar mensagens HTTP para que pudessem então serem capturadas pelo *Wireshark*, estando o servidor e os clientes em computadores distintos.

O servidor Apache é um servidor web compatível com o protocolo HTTP.

Para o envio de pedidos ao servidor usámos um cliente *browser* e outro cliente implementado na linguagem de programação Java, este último estabelecendo uma conexão TCP com o servidor via *socket* onde são mostradas, na aplicação, as mensagens de requisição e de resposta.

Para os dois clientes foram testados vários tipos de mensagens GET obtendo diferentes códigos de resposta por parte do servidor web Apache, demonstrados de seguida.

Para verificar os conceitos teóricos, utilizaram-se duas máquinas, ligadas à mesma rede (neste caso VPN). A Máquina A, de IP 10.10.142.242, é o cliente que faz os pedidos, e a Máquina B, de IP 10.10.138.243, onde estava alojado o servidor pache, que estava a servir as páginas de teste no porto 80.

	IP	Porto
Máquina A	10.10.142.242	54605
Máquina B	10.10.138.243	80
Máquina C	10.10.142.47	80

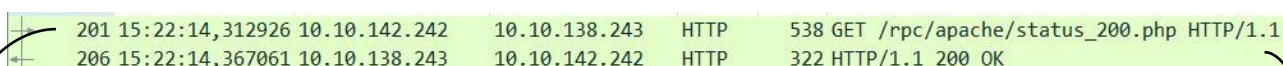
## 2.1 Código 200

Os códigos 2xx representam códigos que indicam que o pedido do cliente foi bem-sucedido. O código mais conhecido é:

- **200 Ok** – Este código representa um standard de resposta de sucesso para pedidos HTTP.

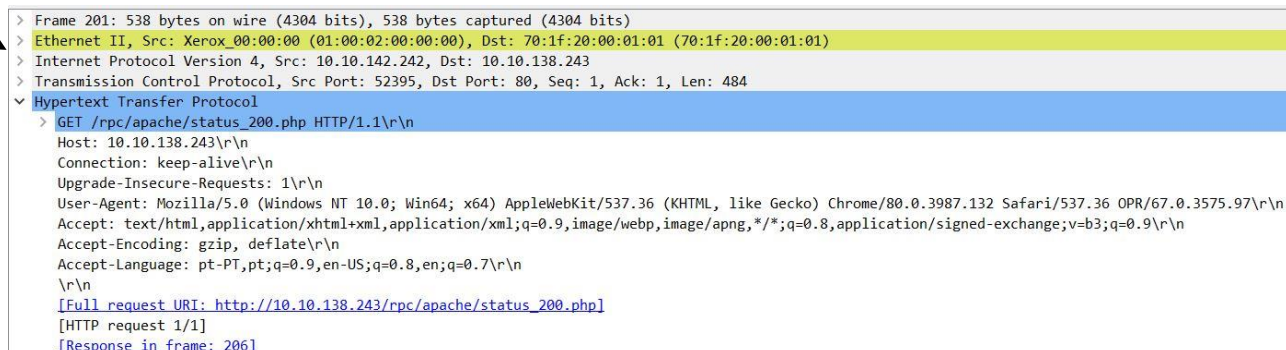
## Resultados Laboratoriais

- Cliente *browser*:



201	15:22:14,312926	10.10.142.242	10.10.138.243	HTTP	538 GET /rpc/apache/status_200.php HTTP/1.1
206	15:22:14,367061	10.10.138.243	10.10.142.242	HTTP	322 HTTP/1.1 200 OK

Figura 6 - Pacotes HTTP capturados do estado 200

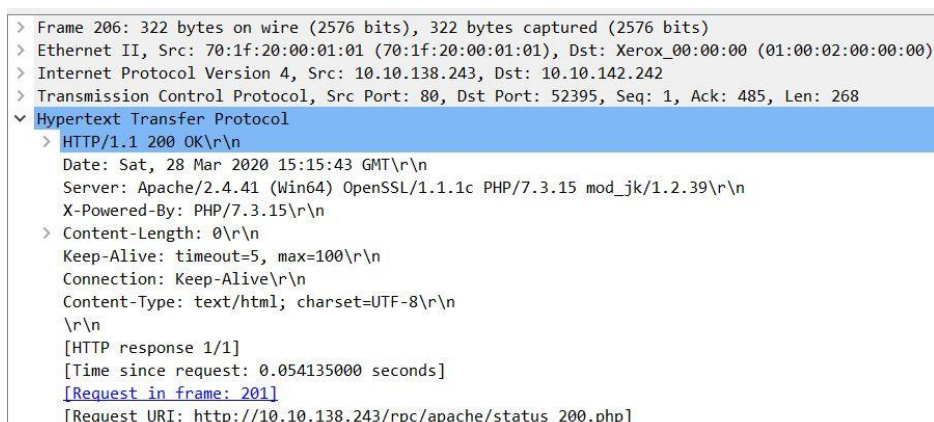


```

> Frame 201: 538 bytes on wire (4304 bits), 538 bytes captured (4304 bits)
> Ethernet II, Src: Xerox_00:00:00 (01:00:02:00:00:00), Dst: 70:1f:20:00:01:01 (70:1f:20:00:01:01)
> Internet Protocol Version 4, Src: 10.10.142.242, Dst: 10.10.138.243
> Transmission Control Protocol, Src Port: 52395, Dst Port: 80, Seq: 1, Ack: 1, Len: 484
v Hypertext Transfer Protocol
  > GET /rpc/apache/status_200.php HTTP/1.1\r\n
    Host: 10.10.138.243\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36 OPR/67.0.3575.97\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7\r\n
    \r\n
    [Full request URI: http://10.10.138.243/rpc/apache/status_200.php]
    [HTTP request 1/1]
    [Response in frame: 206]

```

Figura 7 - Informação do pedido do estado 200, como capturado pelo Wireshark



```

> Frame 206: 322 bytes on wire (2576 bits), 322 bytes captured (2576 bits)
> Ethernet II, Src: 70:1f:20:00:01:01 (70:1f:20:00:01:01), Dst: Xerox_00:00:00 (01:00:02:00:00:00)
> Internet Protocol Version 4, Src: 10.10.138.243, Dst: 10.10.142.242
> Transmission Control Protocol, Src Port: 80, Dst Port: 52395, Seq: 1, Ack: 485, Len: 268
v Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Date: Sat, 28 Mar 2020 15:15:43 GMT\r\n
    Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39\r\n
    X-Powered-By: PHP/7.3.15\r\n
    Content-Length: 0\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.054135000 seconds]
    [Request in frame: 201]
    [Request URI: http://10.10.138.243/rpc/apache/status_200.php]

```

Figura 8 - Informação da resposta do estado 200, como capturado pelo Wireshark

- Cliente Java

```

1890 15:19:09,848133 10.10.138.224 10.10.138.243 HTTP 122 GET /rpc/tp1/apache/status_200.php HTTP/1.1
1899 15:19:09,898904 10.10.138.243 10.10.138.224 HTTP 266 HTTP/1.1 200 OK

```

Figura 9 - Pacotes HTTP capturados do estado 20, com pedidos feitos através do cliente Java

```

> Frame 1890: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
> Ethernet II, Src: 03:00:04:00:00:00 (03:00:04:00:00:00), Dst: 70:1f:20:00:02:01 (70:1f:20:00:02:01)
> Internet Protocol Version 4, Src: 10.10.138.224, Dst: 10.10.138.243
> Transmission Control Protocol, Src Port: 57755, Dst Port: 80, Seq: 1, Ack: 1, Len: 68
> Hypertext Transfer Protocol
  > GET /rpc/tp1/apache/status_200.php HTTP/1.1\r\n
    Host: 10.10.138.243\r\n
    \r\n
    [Full request URI: http://10.10.138.243/rpc/tp1/apache/status_200.php]
    [HTTP request 1/1]
    [Response in frame: 1899]

```

Figura 10 - Informação do pedido do estado 200, como capturado pelo Wireshark, através do cliente Java

```

Message: /rpc/tp1/apache/status_200.php
Request:
GET /rpc/tp1/apache/status_200.php HTTP/1.1
Host: 10.10.138.243

```

Figura 11 - Informação do pedido pelo cliente

```

> Frame 1899: 266 bytes on wire (2128 bits), 266 bytes captured (2128 bits)
> Ethernet II, Src: 70:1f:20:00:02:01 (70:1f:20:00:02:01), Dst: 03:00:04:00:00:00 (03:00:04:00:00:00)
> Internet Protocol Version 4, Src: 10.10.138.243, Dst: 10.10.138.224
> Transmission Control Protocol, Src Port: 80, Dst Port: 57755, Seq: 1, Ack: 69, Len: 212
> Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Date: Sun, 29 Mar 2020 14:12:41 GMT\r\n
    Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39\r\n
    X-Powered-By: PHP/7.3.15\r\n
    Content-Length: 0\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.050771000 seconds]
    [Request in frame: 1890]
    [Request URI: http://10.10.138.243/rpc/tp1/apache/status_200.php]

```

Figura 12 - Informação da resposta do estado 200, como capturado pelo Wireshark, através de pedidos feitos pelo cliente Java

```

Response:
HTTP/1.1 200 OK
Date: Sun, 29 Mar 2020 14:12:41 GMT
Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39
X-Powered-By: PHP/7.3.15
Content-Length: 0
Content-Type: text/html; charset=UTF-8

```

Figura 13 - Informação da resposta data pelo cliente

## 2.2 Código 302

Como já mencionado acima, de um modo geral, todos os códigos de estado entre o 300 e 399 significam redireccionamentos. Os códigos 3XX mais conhecidos são:

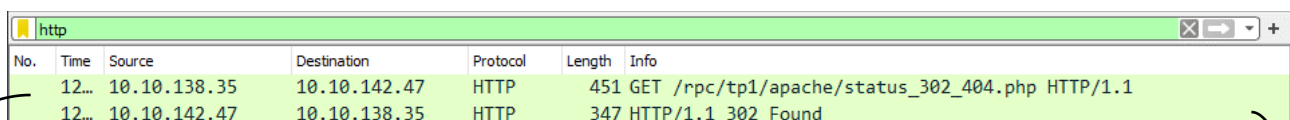
- **301 Moved Permanently** – Quanto um determinado URI foi movido de forma permanente, todos os pedidos serão reencaminhados para a localização determinada pelo servidor
- **304 Not Modified** – Este estado é retornado quando o recurso pedido não foi modificado, então o cliente pode ler da cache onde têm a resposta guardada
- **307 Temporary Redirect** – Este estado existe desde o HTTP/1.1, e informa o cliente que têm de repetir o pedido com outro URI. É também um redireccionamento para outro recurso

De salientar que os estados 301 e 307 estão definidos no RFC 7231 [2], e o estado 302 está definido no RFC 1945 [3].

No nosso caso prático, deparamo-nos com o estado 302 Found (Previously “Moved temporarily”). Este estado informa o cliente para aceder a outro recurso

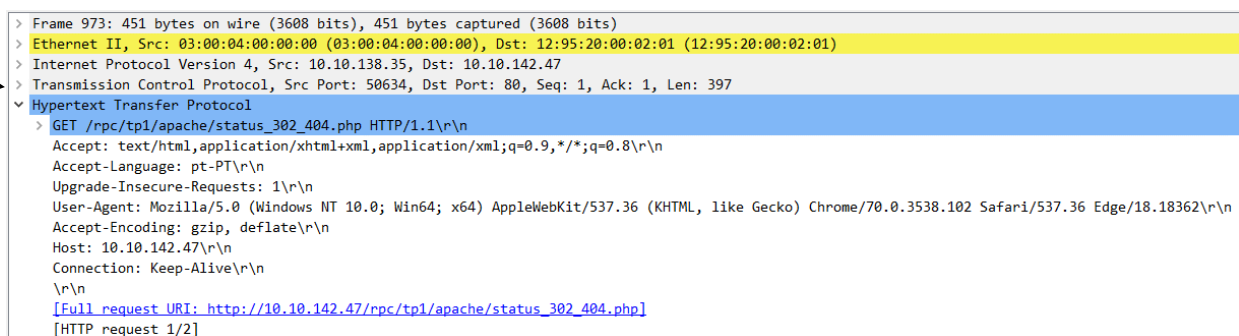
### Resultados Laboratoriais

- Cliente browser



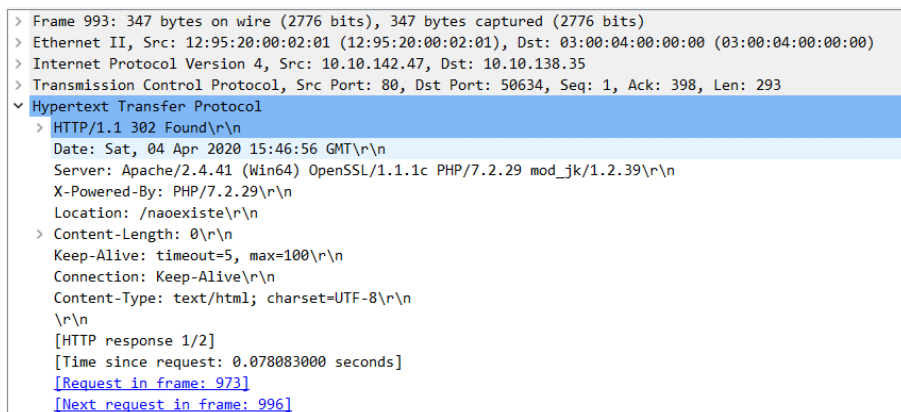
No.	Time	Source	Destination	Protocol	Length	Info
12...		10.10.138.35	10.10.142.47	HTTP	451	GET /rpc/tp1/apache/status_302_404.php HTTP/1.1
12...		10.10.142.47	10.10.138.35	HTTP	347	HTTP/1.1 302 Found

Figura 15 - Pacotes HTTP capturados do estado 302



```
> Frame 973: 451 bytes on wire (3608 bits), 451 bytes captured (3608 bits)
> Ethernet II, Src: 03:00:04:00:00:00 (03:00:04:00:00:00), Dst: 12:95:20:00:02:01 (12:95:20:00:02:01)
> Internet Protocol Version 4, Src: 10.10.138.35, Dst: 10.10.142.47
> Transmission Control Protocol, Src Port: 50634, Dst Port: 80, Seq: 1, Ack: 1, Len: 397
v Hypertext Transfer Protocol
  > GET /rpc/tp1/apache/status_302_404.php HTTP/1.1\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: pt-PT\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18362\r\n
    Accept-Encoding: gzip, deflate\r\n
    Host: 10.10.142.47\r\n
    Connection: Keep-Alive\r\n
    \r\n
    [Full request URI: http://10.10.142.47/rpc/tp1/apache/status_302_404.php]
    [HTTP request 1/2]
```

Figura 14 - Informação do pedido do estado 302, como capturado pelo Wireshark



```
> Frame 993: 347 bytes on wire (2776 bits), 347 bytes captured (2776 bits)
> Ethernet II, Src: 12:95:20:00:02:01 (12:95:20:00:02:01), Dst: 03:00:04:00:00:00 (03:00:04:00:00:00)
> Internet Protocol Version 4, Src: 10.10.142.47, Dst: 10.10.138.35
> Transmission Control Protocol, Src Port: 80, Dst Port: 50634, Seq: 1, Ack: 398, Len: 293
v Hypertext Transfer Protocol
  > HTTP/1.1 302 Found\r\n
    Date: Sat, 04 Apr 2020 15:46:56 GMT\r\n
    Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.2.29 mod_jk/1.2.39\r\n
    X-Powered-By: PHP/7.2.29\r\n
    Location: /naoexiste\r\n
    > Content-Length: 0\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/2]
    [Time since request: 0.078083000 seconds]
    [Request in frame: 973]
    [Next request in frame: 996]
```

Figura 16 - Informação da resposta do estado 200, como capturado pelo Wireshark



- Cliente Java:

10...	10.10.138.224	10.10.138.243	HTTP	126 GET /rpc/tp1/apache/status_302_404.php HTTP/1.1
10...	10.10.138.243	10.10.138.224	HTTP	311 HTTP/1.1 302 Found

Figura 17 - Pacotes HTTP capturados do estado 302, com pedidos feitos através do cliente Java

```
> Frame 19664: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits)
> Ethernet II, Src: 03:00:04:00:00:00 (03:00:04:00:00:00), Dst: 70:1f:20:00:02:01 (70:1f:20:00:02:01)
> Internet Protocol Version 4, Src: 10.10.138.224, Dst: 10.10.138.243
> Transmission Control Protocol, Src Port: 57761, Dst Port: 80, Seq: 1, Ack: 1, Len: 72
> Hypertext Transfer Protocol
  > GET /rpc/tp1/apache/status_302_404.php HTTP/1.1\r\n
    Host: 10.10.138.243\r\n
    \r\n
    [Full request URI: http://10.10.138.243/rpc/tp1/apache/status_302_404.php]
    [HTTP request 1/1]
    [Response in frame: 19674]
```

Figura 18 - Informação do pedido do estado 302, como capturado pelo Wireshark, através do cliente Java

```
Message: /rpc/tp1/apache/status_302_404.php
Request:
GET /rpc/tp1/apache/status_302_404.php HTTP/1.1
Host: 10.10.138.243
```

Figura 19 - Informação do pedido pelo cliente

```
> Frame 19674: 311 bytes on wire (2488 bits), 311 bytes captured (2488 bits)
> Ethernet II, Src: 70:1f:20:00:02:01 (70:1f:20:00:02:01), Dst: 03:00:04:00:00:00 (03:00:04:00:00:00)
> Internet Protocol Version 4, Src: 10.10.138.243, Dst: 10.10.138.224
> Transmission Control Protocol, Src Port: 80, Dst Port: 57761, Seq: 1, Ack: 73, Len: 257
> Hypertext Transfer Protocol
  > HTTP/1.1 302 Found\r\n
    Date: Sun, 29 Mar 2020 14:14:12 GMT\r\n
    Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39\r\n
    X-Powered-By: PHP/7.3.15\r\n
    Location: http://10.10.138.243/naoexiste\r\n
  > Content-Length: 0\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.039446000 seconds]
    [Request in frame: 19664]
    [Request URI: http://10.10.138.243/rpc/tp1/apache/status_302_404.php]
```

Figura 20 - Informação da resposta do estado 302, como capturado pelo Wireshark, através de pedidos feitos pelo cliente Java

```
Response:
HTTP/1.1 302 Found
Date: Sun, 29 Mar 2020 14:14:12 GMT
Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39
X-Powered-By: PHP/7.3.15
Location: http://10.10.138.243/naoexiste
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

Figura 21 - Informação da resposta data pelo cliente

## 2.3 Código 404

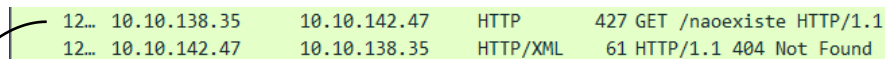
Uma outra classe de códigos de estado que analisámos foi a classe 4xx, que na sua generalidade, significam erros do lado do cliente, como já referido acima. Os mais conhecidos são:

- **400 Bad Request**– Este estado é retornado quando o pedido foi mal construído do cliente.
- **403 Forbidden**– Este estado é retornado quando o cliente não está autorizado a aceder ao recurso que pediu.
- **404 Not Found**– Este estado é retornado quando o recurso que o cliente pediu não está disponível.

Estes estados estão definidos no RFC 7231 [2]. No nosso caso prático, vamos analisar o código de estado 404.

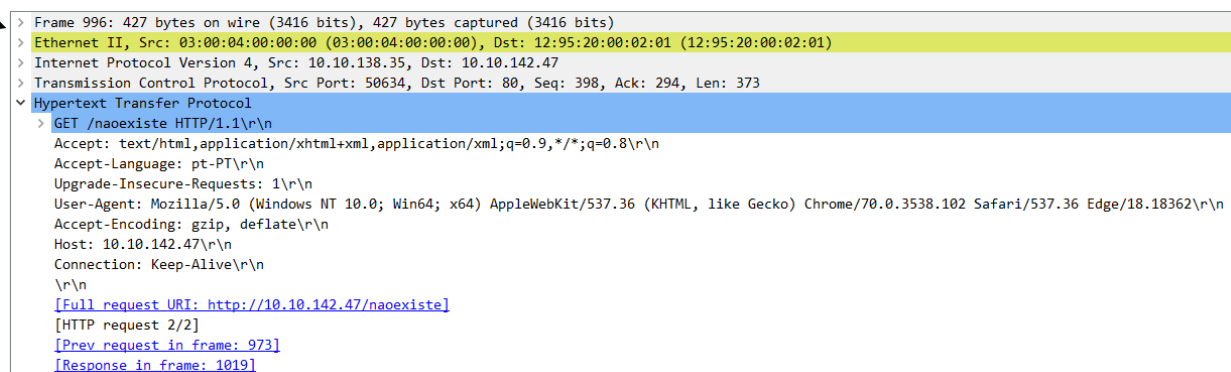
### Resultados Laboratoriais

- Cliente browser



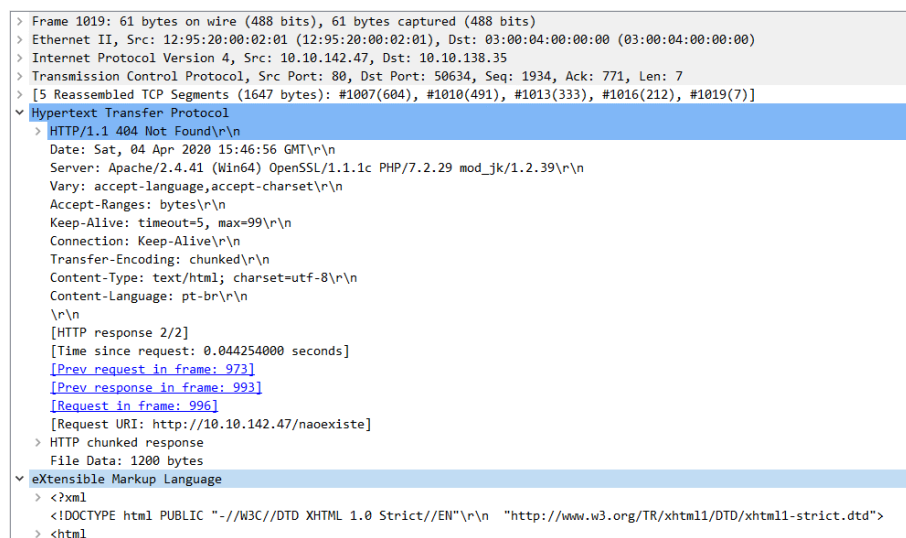
12...	10.10.138.35	10.10.142.47	HTTP	427 GET /naoexiste HTTP/1.1
12...	10.10.142.47	10.10.138.35	HTTP/XML	61 HTTP/1.1 404 Not Found

Figura 22 - Pacotes HTTP capturados do estado 404



```
> Frame 996: 427 bytes on wire (3416 bits), 427 bytes captured (3416 bits)
> Ethernet II, Src: 03:00:04:00:00:00 (03:00:04:00:00:00), Dst: 12:95:20:00:02:01 (12:95:20:00:02:01)
> Internet Protocol Version 4, Src: 10.10.138.35, Dst: 10.10.142.47
> Transmission Control Protocol, Src Port: 50634, Dst Port: 80, Seq: 398, Ack: 294, Len: 373
v Hypertext Transfer Protocol
  > GET /naoexiste HTTP/1.1\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: pt-PT\r\n
  Upgrade-Insecure-Requests: 1\r\n
  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18362\r\n
  Accept-Encoding: gzip, deflate\r\n
  Host: 10.10.142.47\r\n
  Connection: Keep-Alive\r\n
  \r\n
  [Full request URI: http://10.10.142.47/naoexiste]
  [HTTP request 2/2]
  [Prev request in frame: 973]
  [Response in frame: 1019]
```

Figura 23 - Informação do pedido do estado 404, como capturado pelo Wireshark



```
> Frame 1019: 61 bytes on wire (488 bits), 61 bytes captured (488 bits)
> Ethernet II, Src: 12:95:20:00:02:01 (12:95:20:00:02:01), Dst: 03:00:04:00:00:00 (03:00:04:00:00:00)
> Internet Protocol Version 4, Src: 10.10.142.47, Dst: 10.10.138.35
> Transmission Control Protocol, Src Port: 80, Dst Port: 50634, Seq: 1934, Ack: 771, Len: 7
> [5 Reassembled TCP Segments (1647 bytes): #1007(604), #1010(491), #1013(333), #1016(212), #1019(7)]
v Hypertext Transfer Protocol
  > HTTP/1.1 404 Not Found\r\n
  Date: Sat, 04 Apr 2020 15:46:56 GMT\r\n
  Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.2.29 mod_jk/1.2.39\r\n
  Vary: accept-language,accept-charset\r\n
  Accept-Ranges: bytes\r\n
  Keep-Alive: timeout=5, max=99\r\n
  Connection: Keep-Alive\r\n
  Transfer-Encoding: chunked\r\n
  Content-Type: text/html; charset=utf-8\r\n
  Content-Language: pt-br\r\n
  \r\n
  [HTTP response 2/2]
  [Time since request: 0.044254000 seconds]
  [Prev request in frame: 973]
  [Prev response in frame: 993]
  [Request in frame: 996]
  [Request URI: http://10.10.142.47/naoexiste]
  > HTTP chunked response
  File Data: 1200 bytes
v eXtensible Markup Language
  > <?xml
  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  > <html
```

Figura 24 - Informação da resposta do estado 404, como capturado pelo Wireshark

- Cliente Java:

```
16... 10.10.138.224 10.10.138.243 HTTP 106 GET /naoexiste.php HTTP/1.1
16... 10.10.138.243 10.10.138.224 HTTP/XML 63 HTTP/1.1 404 Not Found
```

Figura 25 - Pacotes HTTP capturados do estado 404, com pedidos feitos através do cliente Java

```
> Frame 29397: 106 bytes on wire (848 bits), 106 bytes captured (848 bits)
> Ethernet II, Src: 03:00:04:00:00:00 (03:00:04:00:00:00), Dst: 70:1f:20:00:02:01 (70:1f:20:00:02:01)
> Internet Protocol Version 4, Src: 10.10.138.224, Dst: 10.10.138.243
> Transmission Control Protocol, Src Port: 57767, Dst Port: 80, Seq: 1, Ack: 1, Len: 52
▼ Hypertext Transfer Protocol
  > GET /naoexiste.php HTTP/1.1\r\n
    Host: 10.10.138.243\r\n
    \r\n
    [Full request URI: http://10.10.138.243/naoexiste.php]
    [HTTP request 1/1]
    [Response in frame: 29415]
```

Figura 26 - Informação do pedido do estado 404, como capturado pelo Wireshark, através do cliente Java

```
Message: /naoexiste.php
Request:
GET /naoexiste.php HTTP/1.1
Host: 10.10.138.243
```

Figura 27 - Informação do pedido pelo cliente

```
> Frame 29415: 63 bytes on wire (504 bits), 63 bytes captured (504 bits)
> Ethernet II, Src: 70:1f:20:00:02:01 (70:1f:20:00:02:01), Dst: 03:00:04:00:00:00 (03:00:04:00:00:00)
> Internet Protocol Version 4, Src: 10.10.138.243, Dst: 10.10.138.224
> Transmission Control Protocol, Src Port: 80, Dst Port: 57767, Seq: 1452, Ack: 53, Len: 9
> [5 Reassembled TCP Segments (1460 bytes): #29404(537), #29407(453), #29410(246), #29412(215), #29415(9)]
▼ Hypertext Transfer Protocol
  > HTTP/1.1 404 Not Found\r\n
    Date: Sun, 29 Mar 2020 14:15:14 GMT\r\n
    Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39\r\n
    Vary: accept-language,accept-charset\r\n
    Accept-Ranges: bytes\r\n
    Transfer-Encoding: chunked\r\n
    Content-Type: text/html; charset=utf-8\r\n
    Content-Language: en\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.048007000 seconds]
    [Request in frame: 29397]
    [Request URI: http://10.10.138.243/naoexiste.php]
  > HTTP chunked response
    File Data: 1072 bytes
▼ eXtensible Markup Language
  > <?xml
  > <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
  > <html
```

Figura 28 - Informação da resposta do estado 404, como capturado pelo Wireshark, através de pedidos feitos pelo cliente Java

```
Response:
HTTP/1.1 404 Not Found
Date: Sun, 29 Mar 2020 14:15:14 GMT
Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39
Vary: accept-language,accept-charset
Accept-Ranges: bytes
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
Content-Language: en
```

Figura 29 - Informação da resposta data pelo cliente

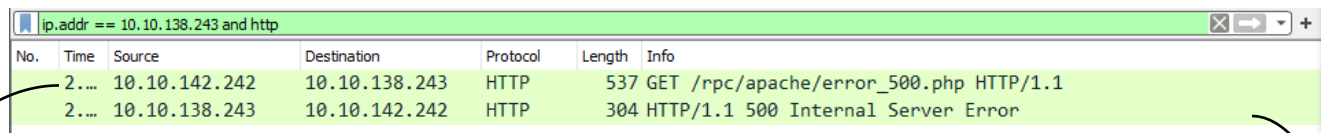
## 2.4 Código 500

Este código 500 representa um código de erro por parte do servidor aquando um pedido válido por parte de um cliente, como descrito na parte introdutória.

- **500 Internal Server Error** – Este erro é retornado quando o servidor simplesmente não consegue responder ao pedido do cliente, e mais nenhum código de erro se adequa.
- **503 Service Unavailable** – Este erro é retornado quando o servidor não consegue aceitar pedidos, porque pode estar sobrecarregado com pedidos de vários clientes
- **504 Gateway Timeout** – Este erro é retornado quando o servidor não consegue responder ao pedido em tempo útil, como por exemplo, no PHP, o *timeout* de um pedido é de 30 segundos (mas é configurável). Se ao fim deste tempo o servidor não der resposta, é retornado um erro destes

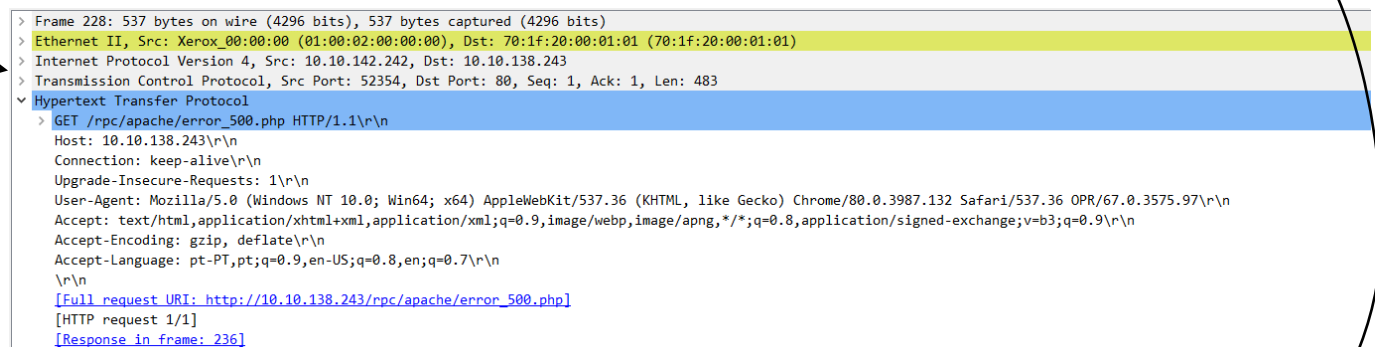
Estes estados estão definidos no RFC 7231 [2]. No nosso caso prático, vamos analisar o código de estado 500.

- Cliente browser



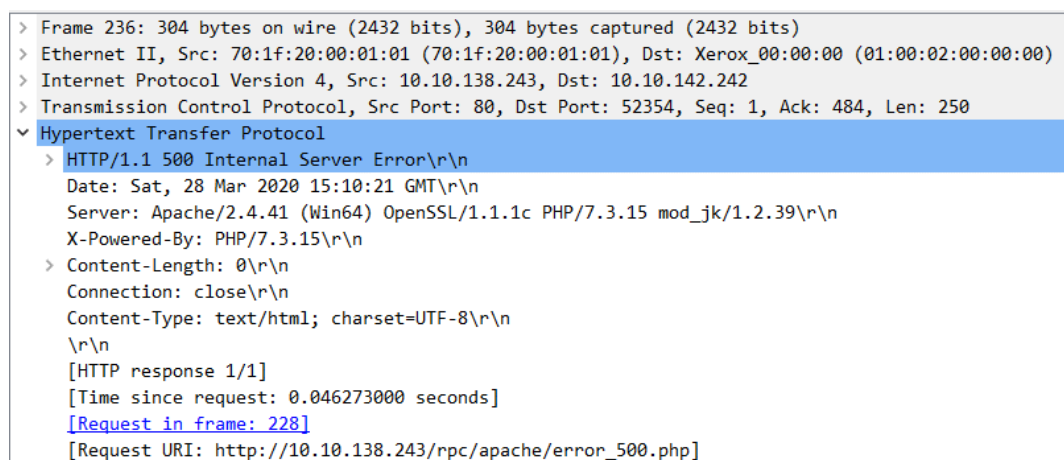
No.	Time	Source	Destination	Protocol	Length	Info
2...		10.10.142.242	10.10.138.243	HTTP	537	GET /rpc/apache/error_500.php HTTP/1.1
2...		10.10.138.243	10.10.142.242	HTTP	304	HTTP/1.1 500 Internal Server Error

Figura 31 - Pacotes HTTP capturados do estado 500



```
> Frame 228: 537 bytes on wire (4296 bits), 537 bytes captured (4296 bits)
> Ethernet II, Src: Xerox_00:00:00 (01:00:02:00:00:00), Dst: 70:1f:20:00:01:01 (70:1f:20:00:01:01)
> Internet Protocol Version 4, Src: 10.10.142.242, Dst: 10.10.138.243
> Transmission Control Protocol, Src Port: 52354, Dst Port: 80, Seq: 1, Ack: 1, Len: 483
v Hypertext Transfer Protocol
  > GET /rpc/apache/error_500.php HTTP/1.1\r\n
    Host: 10.10.138.243\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.132 Safari/537.36 OPR/67.0.3575.97\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
    Accept-Encoding: gzip, deflate\r\n
    Accept-Language: pt-PT;q=0.9,en-US;q=0.8,en;q=0.7\r\n
    \r\n
    [Full request URI: http://10.10.138.243/rpc/apache/error_500.php]
    [HTTP request 1/1]
    [Response in frame: 236]
```

Figura 30 - Informação do pedido do estado 500, como capturado pelo Wireshark



```
> Frame 236: 304 bytes on wire (2432 bits), 304 bytes captured (2432 bits)
> Ethernet II, Src: 70:1f:20:00:01:01 (70:1f:20:00:01:01), Dst: Xerox_00:00:00 (01:00:02:00:00:00)
> Internet Protocol Version 4, Src: 10.10.138.243, Dst: 10.10.142.242
> Transmission Control Protocol, Src Port: 80, Dst Port: 52354, Seq: 1, Ack: 484, Len: 250
v Hypertext Transfer Protocol
  > HTTP/1.1 500 Internal Server Error\r\n
    Date: Sat, 28 Mar 2020 15:10:21 GMT\r\n
    Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39\r\n
    X-Powered-By: PHP/7.3.15\r\n
  > Content-Length: 0\r\n
    Connection: close\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.046273000 seconds]
    [Request in frame: 228]
    [Request URI: http://10.10.138.243/rpc/apache/error_500.php]
```

Figura 32 - Informação da resposta do estado 500, como capturado pelo Wireshark



- Cliente Java:

```

4829 15:19:28,608486 10.10.138.224 10.10.138.243 HTTP 122 GET /rpc/tp1/apache/status_500.php HTTP/1.1
4839 15:19:28,651172 10.10.138.243 10.10.138.224 HTTP 304 HTTP/1.1 500 Internal Server Error

```

Figura 33 - Pacotes HTTP capturados do estado 500, com pedidos feitos através do cliente Java

```

> Frame 4829: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
> Ethernet II, Src: 03:00:04:00:00:00 (03:00:04:00:00:00), Dst: 70:1f:20:00:02:01 (70:1f:20:00:02:01)
> Internet Protocol Version 4, Src: 10.10.138.224, Dst: 10.10.138.243
> Transmission Control Protocol, Src Port: 57759, Dst Port: 80, Seq: 1, Ack: 1, Len: 68
< Hypertext Transfer Protocol
  > GET /rpc/tp1/apache/status_500.php HTTP/1.1\r\n
    Host: 10.10.138.243\r\n
    \r\n
    [Full request URI: http://10.10.138.243/rpc/tp1/apache/status_500.php]
    [HTTP request 1/1]
    [Response in frame: 4839]

```

Figura 34 - Informação do pedido do estado 500, como capturado pelo Wireshark, através do cliente Java

```

Message: /rpc/tp1/apache/status_500.php
Request:
GET /rpc/tp1/apache/status_500.php HTTP/1.1
Host: 10.10.138.243

```

Figura 35 - Informação do pedido pelo cliente

```

> Frame 4839: 304 bytes on wire (2432 bits), 304 bytes captured (2432 bits)
> Ethernet II, Src: 70:1f:20:00:02:01 (70:1f:20:00:02:01), Dst: 03:00:04:00:00:00 (03:00:04:00:00:00)
> Internet Protocol Version 4, Src: 10.10.138.243, Dst: 10.10.138.224
> Transmission Control Protocol, Src Port: 80, Dst Port: 57759, Seq: 1, Ack: 69, Len: 250
< Hypertext Transfer Protocol
  > HTTP/1.1 500 Internal Server Error\r\n
    Date: Sun, 29 Mar 2020 14:13:00 GMT\r\n
    Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39\r\n
    X-Powered-By: PHP/7.3.15\r\n
  > Content-Length: 0\r\n
    Connection: close\r\n
    Content-Type: text/html; charset=UTF-8\r\n
    \r\n
    [HTTP response 1/1]
    [Time since request: 0.042686000 seconds]
    [Request in frame: 4829]
    [Request URI: http://10.10.138.243/rpc/tp1/apache/status_500.php]

```

Figura 37 - Informação da resposta do estado 500, como capturado pelo Wireshark, através de pedidos feitos pelo cliente Java

```

Response:
HTTP/1.1 500 Internal Server Error
Date: Sun, 29 Mar 2020 14:13:00 GMT
Server: Apache/2.4.41 (Win64) OpenSSL/1.1.1c PHP/7.3.15 mod_jk/1.2.39
X-Powered-By: PHP/7.3.15
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8

```

Figura 36 - Informação da resposta data pelo cliente

### 3 – Conclusões

Existem diferenças entre os cabeçalhos dos pedidos por parte do cliente e dos cabeçalhos de resposta por parte do servidor. Normalmente, no cabeçalho dos pedidos dos clientes vão cabeçalhos que descrevem qual o recurso a que o cliente quer ter acesso como também os requisitos e características do cliente, enquanto que nos cabeçalhos de resposta do servidor vão cabeçalhos que descrevem qual o código de sucesso de resposta, as características do recurso pedido, como tamanho e o tipo de dados. Para além destas diferenças também verificámos que o cliente browser coloca mais cabeçalhos do que o cliente implementado por nós. O nosso cliente Java apenas manda os necessários para o protocolo HTTP/1.1, pedido e *host*, enquanto o cliente browser acrescenta mais uns necessários ao bom funcionamento do browser, como por exemplo que linguagens aceita, etc.

A primeira coisa que “salta à vista” é os campos que o *browser* adiciona ao *header*. O campo *User Agent* é, tipicamente, associado a um navegador Web, e historicamente, servia para identificar o *browser* que o utilizador usava. Eventualmente foi copiado por outros *browsers*, tanto que a designação Mozilla 5.0 tornou-se standard. Atualmente, ainda que existam alguns sites que utilizam o *\_User Agent* para identificar o browser (apesar de poder ser facilmente aldrabado), tens vindo a verificar uma migração para uma deteção à base de funcionalidades (determinados *browsers* oferecem funcionalidades que lhe são características) ou também através de prefixação (“webkit”, “moz”, “msie”). Outro *header* é o “*Accept Encoding*”, que diz ao servidor que tipo de codificação de conteúdo o cliente aceita. Neste caso, o *browser* em questão é o Google Chrome, que aceita *gzip* (uma forma de compressão com o algoritmo zip), ou seja, consegue descomprimir ficheiros comprimidos pelo algoritmo *gzip*, o que diminui a *payload* que é descarregada pelo *browser* (e, como consequência, torna a visualização da página mais rápida)

O protocolo HTTP, como dito na parte introdutória, assenta na camada aplicacional e a troca destas mensagens entre servidor e cliente só é conseguida devido à custa da camada de transporte que as transporta entre ambos. Neste caso específico a camada de transporte usada pelo protocolo HTTP é o protocolo TCP, onde o emissor e o recetor conectam-se via *socket* para poderem transmitir os dados.

Dentro dos cabeçalhos de resposta destacam-se os mais comuns como o *Date*, que como valor contém a data e horário de envio da mensagem, o *Server*, com o nome do servidor, o *Content-length*, com o comprimento do corpo da resposta, o *Connection*, com as opções desejadas para a conexão, e o *Content-type*, com o tipo MIME (tipo de mídia de Internet) deste conteúdo.

## 4 – Bibliografia

- [1] “Wireshark,” [Online]. Available: <https://en.wikipedia.org/wiki/Wireshark>. [Acedido em 29 Março 2020].
- [2] “RFC 7231 - HTTP/1.1 Semantics and Content,” Junho 2014. [Online]. Available: <https://tools.ietf.org/html/rfc7231>. [Acedido em 01 Abril 2020].
- [3] “RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0,” Maio 1996. [Online]. Available: <https://tools.ietf.org/html/rfc1945>. [Acedido em 01 Abril 2020].
- [4] “Servidor Apache,” [Online]. Available: [https://pt.wikipedia.org/wiki/Servidor\\_Apache](https://pt.wikipedia.org/wiki/Servidor_Apache).
- [5] “Protocolo HTTP,” [Online]. Available: [https://pt.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol).
- [6] “Mensagem HTTP,” [Online]. Available: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Mensagens>.
- [7] [Online]. Available: <https://www.iperiusbackup.net/pt-br/entendendo-os-conceitos-entre-os-modelos-tcpip-e-osi/>. [Acedido em 29 Março 2020].
- [8] [Online]. Available: [https://pt.wikipedia.org/wiki/Modelo\\_OSI#2\\_-\\_Camada\\_de\\_Liga%C3%A7%C3%A3o\\_de\\_Dados\\_ou\\_Enlace\\_de\\_Dados](https://pt.wikipedia.org/wiki/Modelo_OSI#2_-_Camada_de_Liga%C3%A7%C3%A3o_de_Dados_ou_Enlace_de_Dados). [Acedido em 29 Março 2020].
- [9] [Online]. Available: <https://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-e-o-modelo-osi/>. [Acedido em 29 Março 2020].